Software Stability in the Robotics domain: issues and challenges

Davide Brugali
Dipartimento di Ingegneria Gestionale e dell'Informazione
Università di Bergamo
24044 Dalmine, Italy
brugali@unibg.it

Monica Reggiani
Dipartimento di Ingegneria dell'Informazione
Università di Parma
43100 Parma, Italy
reggiani@ce.unipr.it

***Abstract* - A robot is a physical device that interacts with the external world through sensors and actuators, and carries out tasks autonomously in unstructured, dynamic, partially observable, and uncertain environments. Software applications are the medium to embody intelligence in the robot. Today, most of the research and development of software for robotic systems is based on proprietary design architectures invented from scratch each time. As robots have started to become commodities, a more stability-oriented software development approach is needed. This paper formulates the problem of developing stable software systems in the robotics domain, analyses issues that make the problem hard, and identifies challenges the robotics community has to face in order to build stable software systems.**

Index Terms – Robotics, software stability, domain analysis

I. INTRODUCTION

The development of robotic systems has always been a severe challenge due to the heterogeneous technological issues involved. The process of bringing intelligence to a robot requires strongly tighten capabilities of sensing, processing, and acting. In this scenario, software plays a key role as it is the medium to embody intelligence in the machine.

One of the key attributes of a mature engineering discipline is the routine use of existing solutions in the development of new systems. As robotics systems are becoming more and more complex, distributed, and integrated, there is the need to promote the construction of new systems as composition of reusable building blocks. System modularity and interoperability are key factors that enable the development of reusable software. If a system is modular, its functionality can be customized by replacing individual components. When two or more systems are interoperable, they can be (re)used as components of more complex systems.

Today, most of the research and development of software for robotic systems is based on proprietary design architectures invented from scratch each time. Many valuable robotic applications are monolithic systems that have been developed to solve a specific class of problems.

Usually, reuse is considered only at the implementation stage. This practice limits reuse to fine-grain modules at best and fails to allow for broader utilization of assets at a subsystem or higher level since planning at earlier stages of development is neglected.

This scenario is mainly ascribed to cultural factors. Traditionally Robotics has been the realm of experts in mechanics (robot structures), electronics (sensors, and actuators), automatic control (real-time control), and artificial intelligence (world modelling and task planning). For many years software development has been considered only a sort of side-effect of robotic system construction. The research has always been focused more on inventing new algorithms (e.g. for sensor fusion, motion planning, deliberative control) than on structuring robotic software in such a way to simplify reuse of those algorithms on different robotic platform and in similar applications.

Only during the last few years, the robotic community has started paying attention to advanced software development techniques and methodologies as witnessed for example in [10]. The reason is that robots have now started to become commodities. Intelligent robots help human beings in everyday life [16]; "Web Robotics" is a new fascinating frontier for research and entertainment [9].

Most of these new trends have been made possible by the evolution of the personal computer (PC) (in terms of cost, power, and robustness) and the Internet (in terms of security, speed, and reliability). The Internet is becoming pervasive in those sectors that were out of scope until a few years ago: real-time control [8], telemanipulation and vehicle teleoperation [9], sensor measurement collection and integration for mobile robots and manipulators [42].

In this context, the synergy between Robotics and Software Engineering is strategic. Their mutual benefit is not merely to develop bigger, faster, cheaper software systems, but rather to make it possible to build and evolve new software systems. The aim of this paper is threefold:

1) To formulate the problem of developing stable software systems in the robotics domain.
2) To analyse the issues that make the problem hard
3) To identify the challenges that the robotics community has to face in order to build stable software systems.

The paper is organized as follows. Section I.A identify specific characteristics of robotic software systems and formulates the problem of developing robot software applications. Section I.B reports on related works. Section II analyses the issues and Section III identifies the challenges of building stable software in Robotics. Finally, Section IV draws the relevant conclusions.

A. *Problem Formulation*

Simply stated, a robot (e.g. an industrial manipulator arm or the mobile robot Spirit landed on Mars, or even a humanoid that plays soccer) is a complex hardware-software system made up of sensors (vision system, sonar, laser, etc.), actuators (wheeled platform, grasping tools, etc.), and computational, control, and communication units. It is able to plan and carry out tasks autonomously (e.g. car painting, rock sampling, playing game).

During the execution of its tasks, the robot interacts with the environment (e.g., it grasps an object), with other robots (e.g., it is part of a team that carries a heavy load), and with the human operator (e.g., the plant supervisor).

The physical interaction with the surrounding world occurs through sensors and actuators that are inherently affected by errors, uncertainty, and noise. Robots must deal with unexpected conditions, exhibiting high responsiveness to changes in the real world, i.e. they must react to external stimuli in real-time (e.g. chase the ball in a soccer field).

The collaboration with other robots requires the ability to exchange data (e.g. sensory information acquired from different viewpoints), to coordinate task execution (e.g. assigning roles to robot team-mates), and to synchronize activities (e.g. two robots assembling a work piece).

The collaboration with the human operator is often desirable for complex robotics tasks, such as navigation and manipulation of objects in hazardous environments. Effective Human-Robot interaction [33] requires the exploitation of advanced technologies, such as appropriate graphical user interfaces, speech and gesture recognition tools, and haptic mechanical interfaces

Robot applications are made up of software components (e.g. the motion planner or the map builder) that communicate through a variety of media, such as shared memory (e.g. in the case of on a single robot), wireless LAN or radio links (e.g. in the case of a team of cooperating robots) or even the Internet (e.g. for distributed human-robot interaction).

Common software design problems that every robot system developer has to solve are:

- how to encapsulate common functionality (e.g. an image processing algorithm) in reusable components while preserving real-time performance;
- how to deal with communication, concurrency, and synchronization among heterogeneous subsystems (motion control, sensory data acquisition and elaboration, etc.) usually developed by different project teams or team members;
- how to guarantee graceful degradation of robot functionality in case of software, hardware error or external abnormal condition.
- how to ensure a seamless evolution from the simulation to the final implementation in order to limit the changes to the non-simulated components and therefore to trust the results obtained by the simulation [27].

B. *Related works*

Recently, several initiatives have taken place that aim at defining standards, reference architectures, and middleware for the development of reusable robotic systems:

- The *Robotics Engineering Task Force* [28] is a coalition of industry, academic and government participants. The primary goal of the RETF is to specify reusable, interoperable software technology for mobile robots. The project activity has produced a couple of preliminary documents on APIs and protocols.
- The emerging *Robotics Special Interest Group* [29], supported by the Japan Robot Association, aims at defining a robotics domain architecture, mainly based on OMG's standards. (A few events have been set up which are largely focused on review of state of the art, industry trends, and viewpoints on standardization)
- A few standards have been defined within the military domain. An example is *JAUS* [30], an upper level design for the interfaces of Unmanned Ground Vehicles.

A number of open-source frameworks are now available. They are the result of research projects aiming to support the development of robot systems through the composition of software modules. The following list report a few examples.

- The Player/Stage framework [23] provides the user with Player, a device server that allows the control of a wide variety of robotic sensors and actuators, and Stage, a multiple robot simulator. The robot programmer writes client applications that interact with the Player Server in order to control real or simulated robots.
- The Open Robot Control Software (OROCOS) project [18] is currently developing a CORBA-based component library offering generic functionality for robotics application, such as real-time motion control of robot manipulators and intelligent sensor processing.
- The SmartSoft project [20] sees communications primitives as the core of its robotics component model. The assumption is that reducing the possible component communications to a limited number of patterns can simplify the definition of component interfaces.
- The ORCA project [19] has defined a component model and repository for the domain of mobile robotics. The component model defines specific communication mechanisms and policies among robot components.
- The MARIE project [25] has developed a programming environment that supports the integration of heterogeneous robot control components through the use of application-to-application communication adapters.
- The CARMEN project [24] has designed an open source set of navigation primitives for mobile robot control.

Several other research projects have developed customizable applications used to control a variety of robotic systems.

- The LAAS architecture [22] provides complete support for planning and execution functionality in mobile robots.
- The CLARAty system [21] has been developed at NASA JPL. It consists of a collection of reusable components for planning and scheduling of robot activities.

II. ISSUES IN BUILDING STABLE SOFTWARE IN ROBOTICS

Several techniques have been proposed by the software engineering community to achieve the goal of maximizing the reuse of basic software artifacts, of architectural designs and even of the software designers experience in solving problems in specific contexts, namely Design Patterns, Application Frameworks, and Component Development. Design Patterns are an attempt to overcome the limitation of the pure code reuse of the class library approach by emphasizing the importance of design reuse. Application Frameworks combine code reuse (class library) with design reuse (Design Patterns). A component is a black box building block with a well defined interface and an internal complexity that can range from a simple class to a complete framework.

All of these techniques offer partial (sometimes overlapping) views and solutions to the problem of developing reusable software: individually, they support only one or some parts and phases of the software life cycle.

While a universal reuse solution remains elusive, great improvements can be made by focusing on well-defined areas of knowledge or activity (domains). The term domain is used to denote or group a set of systems (e.g. mobile robots, humanoid robots) or functional areas (motion planning, deliberative control), within systems, that exhibit similar functionality. Domain Engineering is a set of activities aiming at developing reusable artefacts within a domain.

The fundamental tenet of Domain Engineering is that substantial reuse of knowledge, experience, and software artefacts can be achieved by performing some sort of commonality/variability analysis to identify those aspects that are common to all applications within the domain, and those aspects that distinguish one application from another [7].

Recently, the focus of domain engineering has moved from the commonality/variability analysis toward the new concept of stability/changeability analysis. Software stability can be defined as a software system's resilience to changes in the original requirements specification. In [6] the author argues that commonalities analysis identifies domain models and assets which are not necessarily stable. In order to enhance reuse, the software development process should focus on those aspects of the domain that will remain stable over time. Such an approach ensures a stable core design and, thus, stable software artefacts [3].

In order to support stability analysis, the concept of Enduring Business Theme (EBT) has been first introduced in [1] and further investigated in [2, 7, 8] to name a few. An EBT is an aspect of a business domain that is unlikely to change since it is part of the essence of the business. After the appropriate enduring themes of the business are understood and catalogued, a software framework can be constructed that enables the development of business objects (BO) which offer application-specific functionality and support the corresponding EBT [1]. BO have stable interfaces and relationships among each others but are internally implemented on top of more transient components called Industrial Objects (IO) [2].

The identification of EBTs, BOs, and IOs requires a deep knowledge and understanding of the domain. In [5] the author sketches some heuristics for guiding their identification.

The rest of this section analyses three issues in building stable software in Robotics with the goal of supporting the identification of EBTs, BOs, and IOs.

These issues emerge when trying to answer the following questions:

- Which requirements of a robotic system are more likely to remain stable over time?
- Which standards are enforced within the robotic domain?
- What kind of evolution does a robotic system undergo?

The first issue in building stable software in robotics is that *robot systems are highly change-centric systems*.
Robotics is an experimental science that can be analysed from a double perspective.

From one side, it is a discipline that grounds its roots into mechanics, electronics, computer science and cognitive sciences. In their regards, Robotics plays the role of integrator of the most advanced results in order to build complex systems. This means that the stability of robotic artefacts is highly dependant on the evolution of the underlying technologies.

From the other side, Robotics is a research field which pursues ambitious goals, such as the study of intelligent behaviour in artificial systems. Most of its achievements have found applications in industrial settings and everyday life.

Thus, the stable “Robotics business” can be captured by EBTs which express the essence of the artificial systems that Robotics, as engineering discipline, builds for solving concrete problems and which captures the long-term goals that Robotics pursues while developing the robots of the future.

A milestone paper of Rodney Brooks [13] identifies three enduring themes of every robotic system: *situatedness*, *embodiment*, and *intelligence*.

Robot *situatedness* refers to existing in a complex, dynamic, and unstructured environment that strongly affects the robot behavior. For example, the environment is a museum full of people [17] where a mobile robot guides tourists and illustrates masterworks, or a game field where two robot teams play soccer [31], or a manufacturing workcell where an industrial manipulator localizes, and handles work pieces [32].

Robot *embodiment* refers to the consciousness of having a body (a mechanical structure with sensors and actuators) that allows the robot to experience the world directly. The robot receives stimuli from the external world and executes actions that cause changes in the world state.

Robot *intelligence* refers to the ability to express adequate and useful behaviors while interacting with the dynamic environment. For example, the ability of two collaborating robots to transport a load to a target position without colliding with the obstacles might be thought as intelligent behavior.

Clearly these EBTs are expressed at a high level of abstraction and they can be translated into an innumerable variety of concrete robotic systems.

The second issue in building stable software in robotics is that *robot hardware components are highly standardized.*

Robots substantially differ from one other in their mechanical structure. Broadly speaking they can be grouped as follows:

- Industrial robots are manipulator arms that can carry heavy loads (up to 500 Kg), execute highly precise operations (up to 1/10 mm), and perform intensively repetitive tasks (thousands of pick-and-place per hour).
- Mobile robots are autonomous vehicles that can be employed for highly heterogeneous tasks, such as exploring an unknown environment (e.g. the Mars surface), surveying a building, transporting work pieces in an industrial plant. Typical mobility configurations use wheels, tracks, or legs.
- Humanoid robots replicate the structure of human body and are conceived for operating in environments cohabited by people.
- Flying robots, underwater robots, reconfigurable robots and many other variants of the concept of robot are mostly research prototypes.

One of the criteria to recognize EBTs, BOs, and IOs defined in [4] is their *tangibility*. "If an object in a model represents a concrete entity, then it is most likely an Industrial Object" [4]. Tangibility is considered one of the driving factor of instability of IOs. The reason is that physical entities, such as a piece of machinery, are highly sensitive to technological evolutions. Consequently, the software that controls and exposes the hardware should not be at the core of a software system.

According to the *tangibility* criterion, a robot should be classified as Industrial Object. Let's consider the software system that controls a manufacturing workcell for work piece drilling. The drilling operation requires the piece to be positioned in a specific place but it does not matter whether it is handled by a robotic arm or by a humanoid robot. The workcell controller needs to know only when a given piece is ready to be manufactured and that the drilling operation can start. The workcell controller is not affected by the substitution of the robot as far as the software object that interfaces it is insulated from the rest of the system.

Let us now consider the software system that internally controls the robot. Clearly, robot control applications strongly depend on the type of robot used to carry on a task, i.e. the robot mechanical structure determines the requirements of the software applications that controls it. Despite the large variety of robot structures, both commercial and research robots are made up of a limited number of common building blocks. Sensors, motors and actuators, control and processing units, and communication interfaces are quite standardized devices. Factory automation communication protocols have been defined that specify the control interface and the data interchange format of most of these devices (e.g. optical encoders, ultrasonic range sensors, stepping motors, PLC, etc.) in the form of device profiles [37]. Similarly, an intense discussion occurs on open mailing lists [40] in order to gather consensus about the definition of primitive data structures for physical quantities (motion, force, etc.) used in robotics.

Thus, it seems reasonable to argue that the software components wrapping these robotic devices should be quite stable Industrial Objects.

The third issue in building stable software in robotics is that *robot applications build on highly heterogeneous technologies.*

Robotics is perhaps the most inter-disciplinary of engineering endeavours. Building a new robot requires competencies in a variety of disciplines: mechanical engineering (robot structure), electrical engineering (control electronics), electro-mechanical engineering (sensors and actuators), computer engineering (computing hardware), applied mathematics (robot kinematics), statistics (robot and environment models), artificial intelligence (robot skills), software engineering (embedding intelligence into hardware).

From the hardware point of view, Robotics is a quite mature discipline. A limited number of multinational companies build all the manipulator robots employed in the manufacturing industry. Several companies exist that build mobile robots or humanoid robots for research, entertainment, education, and every-day life service [16].

But when it comes to developing the software applications that exploit the robot to carry on useful tasks, only few robotics research groups have the resources and competencies to design and implement, from scratch, the control architectures that embed all the robot functionalities expressing its intelligence.

As stated in Section I.A, the problem is that a robot control application has to satisfy a considerable number of functional and non-functional requirements even for implementing basic robot skills. For example, the ability of navigating in an indoor environment autonomously requires the mobile robot to express several basic behaviors, such as obstacle avoidance, self-localization, path planning, place recognition. The robot has to execute most of these behaviors simultaneously, thus the control application must enforce non-functional requirements such as real-time performance, fault tolerance, concurrence, and distribution.

All of these functionality and requirements are stable aspects of every robotic system (at least for mobile robots) that could be captured by specific Business Objects.

The development of software frameworks based on well designed Business Objects for the robotic domain would be extremely beneficial to the research community since every research group, even the smallest, would have the possibility to concentrate its efforts on a small piece of the robotics puzzle [14]. For example, experts in automated planning could experiment new path planning algorithms for a mobile robot relying on the obstacle avoidance and self localization functionalities encapsulated in off-the-shelf Business Object.

Reuse of consolidated and shared Business Objects allows different teams to test their algorithms on common benchmarks in order to assess performance.

III. CHALLENGES IN BUILDING STABLE SOFTWARE IN ROBOTICS

Robots are ubiquitous [26]. The number of industrial robots is about one million worldwide; 100,000 new robots were installed in 2004. Service robots, carrying out tasks such as milking cows, underwater exploration, and medical assistance, have already reached 21,000 units and the number is set to reach a total of 75,000 by 2007. Expansion of domestic and entertainment market will be even larger, prospecting 6.6 million units sold in the period 2004-2007.

The need of stable software that can be reused with little effort is urgent. Nevertheless, research groups keep solving most of the common design problems from scratch each time a new robotic application is conceived.

The reason can be found in the peculiarity of the robotic domain as emphasized in the previously identified issues: a robot is a multi-purpose (what it is for), multi-form (how it is structured), and multi-function (what it is able to do) machine. Thus, the goal of defining a one-fit-all architecture or framework for every robotic application is elusive.

While in some application domains, such as telecommunications [36], factory automation [34], enterprise information systems [35], large companies or international committees have defined "de iure" standards for reference architectures and frameworks, we argue that this is not a viable approach for solving the problem of developing reusable and interoperable robotic system.

In contrast, we believe that, if the robotics community shares the same vision of stability-oriented software development, stable software solutions can naturally emerge from the profitable exchange of knowledge and experience among experts in the robotics domains and from common practice. In order to pursue this result we have identified at least three challenges that can be faced by the robotics community as a whole with profitable contributions coming from even smallest research group.

The first challenge in building stable software in robotics is the *assessment of a domain model in terms of Enduring Business Themes*.

Simply stated, the business of robotics research consists in "designing algorithms that allow robots to function autonomously in unstructured, dynamic, partially observable, and uncertain environments" [38]. This is a quite ambitious goal that describes the "user requirements" of every robotic system. These requirements are commonly expressed in fuzzy and ambiguous terms that make it hard to understand what the functionality needs to be in the robotic system being built.

A big challenge for the robotic community is the definition of a common language as a collection of EBTs, to express robotic requirements and functionality in order to build comparable and reusable robotic systems. In section II, we have identified three robotic enduring themes: *situatedness*, *embodiment*, and *intelligence*. More specific EBTs can be derived by breaking off these highly abstract concepts following a top-down identification strategy.

For example, *situatedness* (property of being immersed within the real world) can be specialized in *robot localization* (where am I?) and *object recognition* (what is this?). Robots, indeed, show the stable properties of knowing where they are and recognizing objects and places already visited.

Similarly, *embodiment* (consciousness of having a body) can be detailed in *proprioception*, i.e. the ability to perceive its own state in terms of relative position, orientation, and movement of the robot's body and its parts, and *actuation*, i.e. the ability of changing its internal state and the interaction with the external world.

More elusive is the concept of *intelligence* (the ability of expressing useful behavior). It can be better described in terms of more specific EBTs, such as *deliberativeness*, i.e. the ability of planning and revising future actions in order to achieve a given goal while taking into account the mutable conditions of the external environment, and *adaptability*, i.e. the ability of changing its behaviour in response to external stimuli according to past interactions with the real world.

The second challenge in building stable software in robotics is the *identification of middleware functionalities to interconnect robotic Industrial Objects*.

In Section II we have pointed out that, from a mechanical point of view, robotic systems are heterogeneous compositions of quite standard devices and that application requirements strongly depend on the robot's structure. For example, a video camera mounted on a mobile robot can be used for site surveillance, while the same video camera mounted on a room ceiling can help the robot to self-localize.

Easy peripheral integration, universal access to shared data and resources, and inexpensive reconfiguration of control applications are key factors that enable software stability in the robotics domain. There are standardized computer buses, such as ISA Plug and Play and PCI, standardized I/O ports such as RS-232 and Ethernet, and standardized software operating systems such as Windows, Linux, and VxWorks. There is the need to leverage plug-and-play integration to the Industrial Object level. This goal has been achieved in the factory automation domain by big companies and international consortia. For example, OPC [34] provides interoperability between field devices, control systems, and enterprise-wide business applications. Similarly, Industrial IT [41] defines the concept of Aspect Object to represent, manipulate, integrate, and access in real-time information and data of plant devices.

Several robotics projects, such as those documented in [19, 20, 25], aim at creating integrated software environments for the development of reusable objects which are interoperable across robotic platform and control applications. What is still missing to achieve stability of middleware service for the robotics domain (those that sit between physical devices and control applications to enable interoperation) is a clear understanding of the functional and communication requirements of robotic Industrial Objects. The challenge is to let emerge this knowledge form common practice.

The third challenge in building stable software in robotics is the *documentation of architectural solutions to recurrent design problems emerging from the robotic domain which can be captured by reusable Business Objects*.

In Section I.B we have reported on a number of initiatives aiming at developing reusable software frameworks encompassing reference architectures, code libraries and middleware infrastructures. Most of these development efforts have produced software assets that often solve similar problems (e.g. abstracting robot devices) but with emphasis on different aspects (e.g. simulation of physical devices or interoperability among heterogeneous components).

Using a framework raises some interesting questions. When is it more convenient to adopt an existing framework or build everything from scratch? Which are the criteria to select the right framework? What is the process to transform a framework into a concrete application?

Answering these questions requires the application developer to compare the specifications of existing frameworks against the requirements of the new application.

The key to enable the entire reusability of a framework is its documentation: it must be clear what the framework offers, which design problems it solves, and to what extent it is customizable. In a pioneering paper [12] Johnson argues that design patterns document frameworks and help to ensure the correct use of framework functionality. Patterns are a design documentation technique, which communicates the reason of a design decision, not only the result. However, individual patterns may be considered just recipes of a cookbook, while the overall design of a framework is better documented by a corresponding pattern language [15].

In order to make the large software corpus available today within the Robotics community reusable, there is the need to make the domain knowledge and design experience behind it transparent to the developers of new robotic systems.

We argue that this can be achieved if the robotics community pursues three main objectives:

- to identify recurrent problems in the current practice of software development in robotics;
- to formulate these problems in a common language that can allow the understanding of the proposed solutions and make them reusable for the resolution of new problems;
- to share recurrent problems and reusable solution within the robotics community.

IV. Conclusions

This paper has analyzed principles and practice of software development in Robotics. It has pointed out the difficulties and the opportunities to let a stability-oriented approach emerge from the current practice. Three strategic challenges for the robotics community have been identified:

- share a common language to express application requirements in the form of EBTs;
- identify common requirements for IOs interconnection;
- document design solutions that can be captured by BOs.

References

[1] Cline, M., Mike G., Howard Y. “Enduring business themes”, sidebar in Building Application Frameworks: Object-Oriented Foundations of Framework Design, M. Fayad et al. Eds, John Wiley and Sons, 1999.

[2] Clien M., Girou M., “Enduring Business Themes”. CACM, Vol 43(5), May 2000 pp. 101-106

[3] Fayad M.E., Altman A., “An Introduction to Software Stability”, CACM Vol. 44(9), September 2001

[4] Fayad M.E., “Accomplishing Software Stability”, CACM 45(1),2002

[5] Fayad M.E., “How to Deal with Software Stability”, CACM 45(4), 2002

[6] Haitham S. Hamza, “SODA: A Stability-Oriented Domain Analysis Method”, OOPSLA’04, Oct. 24–28, 2004, Vancouver, Canada.

[7] Coplien J., Hoffman D., Weiss D., “Commonality and Variability in Software Engineering”, IEEE Software, Vol. 15(6), 1998

[8] I. Elhajj, N. Xi, and Y.-H. Liu, “Real-time control of internet based teleoperation with force reflection,” in Proc. 2000 IEEE Int. Conf. Robotic and Automation, ICRA, 2000, pp. 3284–3289.

[9] Siegwart R., Goldberg K., (Eds.), “Beyond webcams, An Introduction to Online Robots", MIT Press, Cambridge, MA, 2002.

[10] Brugali D., Fayad M.E., Menga G., Volz R. (Eds.) Special Issue on “Object-Oriented Methods for Distributed Control Architectures”, IEEE Transactions on Robotics and Automation, VOL. 18(4), August 2002

[11] Hattig M., Horswill I., Butler J., “Roadmap for Mobile Robot Specifications”, Proceedings of the 2003 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems Las Vegas, Nevada · October 2003

[12] Johnson, R. 1992. Documenting frameworks using patterns. In Proceedings of OOPSLA'92 (October 1992).

[13] Brooks R.A., Intelligence Without Reason, Proceedings of the 12th Int. Joint Conference on Artificial Intelligence, Sidney 1991

[14] Kortenkamp D., Schultz A.C., „Integrating Robotics Research”, Autonomous Robots 6, 243–245, 1999 Kluwer Academic Publishers.

[15] Brugali D., Menga G. “Frameworks and Pattern Languages: an Intriguing Relationship”, ACM Computing Surveys, 2000

[16] IEEE RAS TC on Service Robotics. http://www.service-robots.org

[17] Burgard W, Cremers A, Fox D et al., “Experiences with an interactive museum tourguide robot”. Artificial Intelligence 114: 32-149, 2000

[18] OROCOS http://www.orocos.org

[19] ORCA http://orca-robotics.sourceforge.net/index.html

[20] SmartSoft http://www.rz.fh-ulm.de/~cschlege/orocos/index.html

[21] CLARAty http://claraty.jpl.nasa.gov/

[22] Open Software for Autonomous Systems http://softs.laas.fr/openrobots/

[23] Player/Stage http://playerstage.sourceforge.net

[24] CARMEN http://www-2.cs.cmu.edu/~carmen/

[25] MARIE http://marie.sourceforge.net/

[26] UNECE, http://www.unece.org/press/pr2004/04robots_index.htm

[27] Aarsten A., Brugali B. and Menga G. “Designing Concurrent and Distributed Cotnrol Systems“, CACM Vol. 39 N. 10, October 1996

[28] Robotics Engineering Task Force http://www.robo-etf.org

[29] Robotics Domain Special Interest Group, http://www.omg.org/news/releases/pr2005/02-17-05.htm

[30] *JAUS* http://www.jauswg.org

[31] Kim, J.-H., Kim, D.-H., Kim, Y.-J., Seow, K.-T., “Soccer Robotics” Springer Tracts in Advanced Robotics, Vol. 11 2004

[32] Bettini A. et al., Vision-Assisted Control for Manipulation Using Virtual Fixtures, IEEE Trans. on Robotics and Automation, VOL. 20(6), 2004

[33] Prassler, E. et al. (Eds.) “Advances in Human-Robot Interaction” Springer Tracts in Advanced Robotics, Vol. 14, 2005

[34] OPC Foundation, www.opcfoundation.org

[35] Johnson V., “The San Francisco project: business process components and infrastructure”, ACM Computing Surveys, Vol 32(1es), 2000

[36] TINA Consortium, http://www.tinac.com/

[37] PROFIBUS standard EN 50170, http://www.profibus.com/

[38] Gaurav S. Sukhatme, Maja J. Mataric, “Robots: intelligence, versatility, adaptivity” Communications of the ACM , Vol 45(3), 2002, pp. 30 - 32

[39] Pfeifer R., Scheier C., Understanding Intelligence. 1999 MIT Press

[40] BROS-I http://sourceforge.net/mailarchive/forum.php

[41] ABB Industrial IT http://www.abb.com

[42] Makarenko A. et al., "A Decentralized Architecture for Active Sensor Networks". ICRA2004, April 26 - May 1, New Orleans, LA, USA